A New Approach to On-demand Loop-Free Multipath Routing

Jyoti Raju
jyoti@cse.ucsc.edu
Computer Science Dept.
University of California
Santa Cruz, CA 95064

J.J. Garcia-Luna-Aceves
jj@cse.ucsc.edu
Computer Engineering Department
University of California
Santa Cruz, CA 95064
Networking and Security Center
Sun Microsystems Laboratories
Palo Alto, California 94303

Abstract— **We present and verify ROAM, an on-demand routing algorithm that maintains multiple loop-free paths to destinations. Each router maintains entries only for those destinations for which data flows through the router, which reduces storage space requirements and the amount of bandwidth needed to maintain correct routing tables. In ROAM, routes are established and maintained on demand using diffusing computations. A router does not send updates for active destinations, unless its distance to them increases beyond a given threshold. ROAM maintains state that informs routers when a destination is unreachable and prevents routers from sending unnecessary search packets attempting to find paths to an unreachable destination. ROAM is shown to converge in a finite time after an arbitrary sequence of topological changes and is shown to be loop-free at every instant. The time and communication complexities of ROAM are analyzed.**

Keywords— **On-demand, Loop-Free Routing, Distance vector routing**

I. Introduction

On-demand routing algorithms were originally proposed for mobile ad-hoc networks, which are autonomous systems of mobile hosts connected by wireless links. Ad-hoc radio networks containing mobile nodes suffer from a limited amount of bandwidth, and one way to reduce the amount of bandwidth consumed is to maintain routes to only those destinations for which a router has data traffic. However, as the use of the Internet increases, we can foresee Internet-supported enterprises in which all business activities are conducted via the Internet. Financial services, securities exchanges and emergency services are examples of applications that will require reliable Internet connectivity. In such situations, it is not unusual for organizations to provide topological redundancy in the form of multiple links with separate gateway routers to the Internet. One issue with using multiple egress links is that manual configuration of internal routers would be needed to make the default route point to one of the gateway routers. This would require considerable planning and monitoring. Running an on-demand routing protocol in the routers of the network would allow routers to dynamically pick different gateway routers for different destinations in the Internet. This would provide implicit load balancing, because some gateway routers can offer better paths to certain destinations. Also, the routers would transition smoothly to any available gateway router if the currently used link to the Internet failed. The main advantage of on-demand routing over the table-driven routing approach would be that internal routers would have to maintain routes only for the subset of routes they are using. The flood search used by on-demand routing would only be propagated up to the edge of the organization network. This mechanism can be used to maintain routes to both internal destination and external destinations.

This work was supported in part at UCSC by the Defense Advanced Research Projects Agency (DARPA) under grant F30602-97-2-0338.

All prior work in on-demand routing has focused on wireless networks and has followed three main approaches to ensuring that the routes obtained are free of long-term loops. All the protocols use a flood search to create routes to destinations. The dynamic source routing (DSR) protocol [6] is an example of using complete-path information in the data packets to avoid loops. The ad-hoc on-demand distance vector routing algorithm (AODV)[10] is an example of using sequence numbers to avoid long-term loops. In AODV, each destination maintains a sequence number that it updates every time there is a connectivity change with its neighbors. The Temporally-ordered routing algorithm (TORA) [9] is an example of using time stamps and internodal coordination to avoid looping. TORA uses a link-reversal algorithm [2] to maintain loop-free multipaths that are created by a query-reply process similar to the above two algorithms.

This paper introduces a new approach to the establishment and maintenance of loop-free routes on demand in either wireless networks or wired networks. We present the ROAM (routing on-demand acyclic multipath) algorithm, which uses internodal coordination along directed acyclic subgraphs defined solely on the routers' distances to destinations. We call the operations used to coordinate nodes "diffusing computations". ROAM extends the diffusing update algorithm (DUAL) [3] to provide routing on demand.

We observe that, with some of today's on-demand routing protocols, when a destination fails or becomes unreachable from a network component, a source trying to obtain a path to the destination finds that its flood-search for the destination fails, but is unable to determine whether or not it should start another flood search, which could have failed simply due to temporary link failures induced by fading or node mobility, for example. There are no inherent mechanisms in these on-demand routing protocols that would prevent a source from repeating its search in the event that the destination is not reachable, which we call the *searching-to-infinity* problem. This problem also makes the (wired or wireless) network running an on-demand routing protocol susceptible to a unique form of attack, where a malicious router can indefinitely query a network for a destination that does not exist, thus causing congestion due to queries. Consequently, external mechanisms are used today in order to stop sources from sending unnecessary queries. In DSR and AODV, routers do not keep state about the search queries in progress, and the application accessing the on-demand routing service must implement a hold-down time after a search fails; however, just as it was the case in Cisco's IGRP [4], it is difficult to determine an adequate length of hold-down time or how many times a source should persist requesting a path to a destination. In addition, each source must go through the process independently. On the other hand, in TORA, routers that have processed a search query keep the state and the source need not repeat the search query multiple times. This is an advantage over stateless routing protocols that we further improve in ROAM, where a search query in a connected component results in either the source requesting a route to a destination obtaining its answer or all the routers determining that the destination is unreachable. Hence, ROAM

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
1999		**00-00-1999 to 00-00-1999**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
A New Approach to On-demand Loop-Free Multipath Routing	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California at Santa Cruz,Department of Computer Engineering,Santa Cruz,CA,95064**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**6**	

eliminates the need for application-level mechanisms to prevent excessive flooding of searches in the event destinations are not reachable.

Section II describes the notation used in this paper to describe ROAM. Section III describes ROAM in detail. Section IV addresses loop freedom in ROAM; the complete proof of loop-freedom is not presented here due to space limitations. Section V addresses ROAM's complexity and shows that ROAM achieves the same type of performance as other alternatives, while preventing the search-to-infinity problem. Finally, Section VI presents our conclusions.

II. Network Model and Notation

To describe ROAM, we model a network as an undirected graph $G(V, E)$. V is the set of routers in the network and E is the set of links in the network. Each router has a unique ID and a link is said to exist between two routers if they can exchange packets. Each link has two costs associated with it - one in either direction.

Our description and verification of ROAM assumes the existence of a link level protocol ensuring that:

- ROAM is notified about the existence of a new neighbor or the loss of connectivity with a neighbor within a finite time.
- Link costs are always positive and a failed link has infinite cost.
- All control packets are sent reliably and are received within a finite amount of time. If the packets cannot be sent after a specified amount of retries, then the link layer marks the neighbor as being down and sends an indication to the routing protocol. Since control packets travel only one-hop, we only require single hop reliability.
- All messages and changes in the cost of links and the addition and deletion of neighbors are processed within a finite time.
- Messages can be transmitted over a link only when the link is perceived as being up.

Reliable message transmission can be easily added into a routing protocol for a wired network [8]. In a wireless network, the logical link control [1] necessary to satisfy the above assumptions can be implemented on top of any MAC protocol designed for wireless links based on collision avoidance (e.g., IEEE802.11), TDMA, or any of the various dynamic scheduling MAC protocols proposed recently [12], [7], [13] without requiring additional network-level control packets.

The following notation is used throughout this paper:

N: the set of destinations a router knows about.

N_i: the set of routers connected through a link with router i, i.e., the set of neighbors of router i.

l_k^i: the cost of the link to neighbor k; the cost of a failed link is assumed to be ∞.

$D_j^i(t)$: the current distance maintained by router i for destination j at time t.

$D_{jk}^i(t)$: the distance from neighbor k to router j as known by router i at time t.

$FD_j^i(t)$: the *feasible distance* at router i for destination j; this distance is used to check if the *feasibility condition* (defined in Section III-B) is satisfied.

$RD_j^i(t)$: the distance to destination j used in messages sent to the neighbors at time t.

$D_j^{*i}(t)$: the smallest value assigned to D_j^i from the time i became passive up to time t.

SS_j^i: is the set of neighbors of router i that offer loop-free routes to destination j; any neighbor k whose distance as known by i, D_{jk}^i is lesser than the feasible distance FD_j^i belongs to this set.

s_j^i: the *successor* for destination j; this successor offers a loop-free path to destination j and is used for data packets.

o_j^i: the *query origin flag* records how a router gets into the active state (further explanation in section III-D).

T_j^i: this timestamp is maintained for each destination. It indicates the last time a data packet was seen for the destination.

$ST_{jk}^i(t)$: this value can be set to active or passive; when set to active, it indicates that router i has sent a query to neighbor k and expects it to return a reply for destination j.

III. ROAM

A. Information stored and exchanged by each router

Each router maintains a *distance table*, a *routing table* and a *link-cost table*. The distance table at router i is a matrix containing for each destination j and for each neighbor k of router i, the distance D_{jk}^i as last reported by k and a reply status flag ST_{jk}^i. The routing table at router i is a column vector containing, for each destination j, the distance to the destination D_j^i, the feasible distance FD_j^i, the reported distance RD_j^i, the successor s_j^i, the query origin flag o_j^i and the timestamp T_j^i. The link-cost table lists the costs of links to each known adjacent neighbor (l_k^i).

There are three types of control packets used by the routing protocol: queries, replies and updates. A control packet from router i to router k contains the addresses i and k and the address of the destination j for which a path is desired. The packet also contains a field indicating the reported distance (RD_j^i) from router i to destination j. A flag u_j^i indicates whether a control packet is an update, a query or a reply to a query. The distance in a packet can be set to any positive value including infinity.

B. Active and Passive States in ROAM

A router i updates its routing table for a destination j when: (a) it needs to add an entry for j, (b) it needs to modify its distance to j (including setting that distance to ∞), and (c) it decides to erase the entry for j.

For a given destination, a router that has sent queries to all its neighbors and is waiting for replies from at least one of its neighbors is said to be active; otherwise, it is said to be passive. With respect to a given destination j, a router running ROAM can be in one of the following three states: (a) passive knowing or not knowing about j's existence, (b) active waiting to obtain distance information about j (while creating routes), and (c) active waiting for replies from neighbors about a known destination j (while maintaining routes). A router i initializes itself in the passive state with a distance of zero to itself ($D_i^i = FD_i^i = RD_i^i = 0, s_i^i = i, T_i^i$ =present time).

To maintain loop-free routes, each router can only pick as successor a neighbor that satisfies either of the two feasibility conditions. To remain passive and have a loop-free route, a router needs to have a neighbor that is a *feasible successor* (fs_j^i). The feasible successor provides the shortest loop-free path to the destination. The passive feasibility condition (*PFC*) is to be satisfied by a router's successor when a router is passive. The active feasibility condition (*AFC*) comes into play only when a router is active, i.e., there is no longer any feasible successor. All neighbors in SS_j^i satisfy AFC, which implies that they provide loop-free paths. However, when a router is active, SS_j^i no longer contains the feasible successor.

PFC: If at time t router i needs to change its successor, it can choose as its new successor any neighbor $q \in N_i(t)$ for which $D_{jq}^i(t) + l_q^i(t) = Min\{D_{jx}^i(t) + l_x^i(t)\ \forall x \in N_i(t)\}$ and $D_{jq}^i(t) < FD_j^i(t)$, where $FD_j^i(t) = D_j^{*i}(t)$.

AFC: If at time t router i becomes active, then it can set its successor to any neighbor $q \in N_i(t)$ where $D_{jq}^i(t) < FD_j^i(t)$. If there is no such router, then the router maintains its earlier successor until it becomes passive again.

The feasible successor plays a key role in maintaining loop-freedom, because it creates a total ordering of distances along any path [3]. Only the distance through the feasible successor is reported in control messages. Therefore, we are able to maintain multiple routes while introducing no extra latency or control messages. Neighbor routers that

satisfy AFC and not PFC can be used for forwarding packets even while the router is active or passive, but their distances are not used in path calculations.

Consider Fig. 1 in which router j is the destination and routers a, b and c are neighbors of router i. Router b satisfies PFC and therefore is the successor and feasible successor of router i; router a is in the successor set SS_j^i as it satisfies AFC. If link (i, b) fails, router a is marked as successor, even though router c offers a shorter path. This is done because we know that only router a guarantees a loop-free path. However, because the path through a is not the shortest possible, router i becomes active and start a diffusing computation. The rest of this

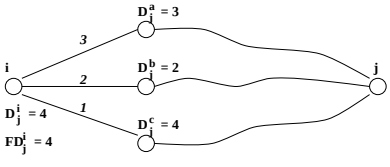


Fig. 1. Successors in ROAM

section describes how diffusing computations are used in ROAM to create, maintain, and delete routes to destinations on demand.

C. Creating Routes

When a router gets a data packet for a destination for which it has no entry in its routing table, it starts a *diffusing search*, which is a diffusing computation originated by a source and propagated by each router that has no entry for the destination. The source of this search can be either the source of the data packet or any intermediate router on the path from the source to the destination. The diffusing search propagates from the source out on a hop by hop basis, until it reaches a router that has an entry for the requested destination, in which case the router replies with its distance to it. At the end of the search, the source either obtains a finite distance to the destination or all the nodes in the same connected component determine that the destination is unreachable ($D_j^i = \infty$ and node is passive).

A router starting the diffusing search adds the destination to its routing table ($D_j^i = FD_j^i = RD_j^i = \infty$, $s_j^i = null$, $o_j^i = 1$,T_j^i = present time) and distance table ($D_{jk}^i = \infty \ \forall k \in N_i$), becomes active for the destination (ST_{jk}^i =active $\forall k \in N_i$) and sends a query to its neighbors. The queries used in a diffusing search report a distance $RD_j^i = \infty$.

A neighbor i that receives a query for j and has no entry for the destination adds the destination to its routing table ($D_j^i = FD_j^i = RD_j^i = \infty$, $s_j^i = null$, $o_j^i = 3$, T_j^i = present time) and distance table ($D_{jk}^i = \infty \ \forall k \in N_i$), becomes active for the destination (ST_{jk}^i =active $\forall k \in N_i$) and forwards the query to its neighbors.

Replies to a query can result in making active routers passive and therefore shrinking the diffusing search and finally ending it. When a router gets a reply from neighbor k, it records the reported distance ($D_{jk}^i = RD_j^k$) and resets the active flag (ST_{jk}^i =passive).

Replies are sent by routers when any of the following three conditions are satisfied:

1. A router that already has an entry for the destination, infinite or finite, sends back a reply immediately with $RD_j^i = D_j^i$, because PFC is satisfied already. This condition also holds for the destination of the diffusing computation.
2. A router that is already active for the destination sends a reply back immediately with $RD_j^i = D_j^i$.
3. A router i other than the source of the diffusing search that has received replies from all its neighbors sends a reply with $RD_j^i = D_j^i$. Before sending the reply, i sets ST_{jk}^i =passive for all $k \in N_i$ and sets its feasible distance, reported distance and distance to the minimum value of $D_{jk}^i + l_k^i$ for all $k \in N_i$. The neighbor that offers the minimum value becomes the new successor and feasible successor. If all the reply distance values received by a router are infinity, then the router sends a reply with $RD_j^i = \infty$ to the neighbor that sent it the query.

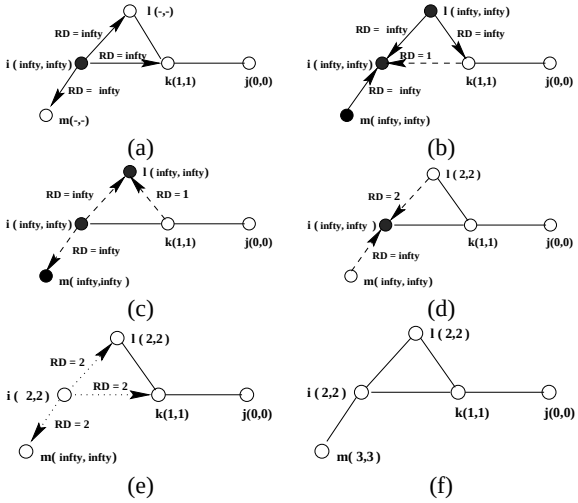


Fig. 2. Creating routes: Router i searches for destination j

Fig. 2 shows a diffusing computation where router i is searching for a path to router j. For simplicity, all link costs are assumed to be 1. The first entry in the parenthesis is the distance to destination j and the second entry is the feasible distance to destination j. Routers k and j are the only ones who know of router j's existence. The queries are denoted by arrows with solid lines. The arrows with dashed lines are the replies and the dotted arrows are updates. Black circles are routers that are active and white circles are routers that are passive.

If the source router gets a finite distance after a search, there can exist certain areas of the network that did not receive replies confirming the existence of the destination. In Fig. 2, router m would correspond to such a router. These routers would assume that they are partitioned from the destination because they still have a distance of infinity to the destination. To avoid this condition, we incorporate a mechanism called *threshold updates.* These updates are sent by a router when its distance to a certain destination changes by more than a defined threshold ΔD. The parts of the network that have infinite entries for a destination that is not partitioned eventually change their distances to the correct distance. Routers that have no entry for the destination do not propagate updates.

D. Handling Link Cost Changes

Link cost changes to a router k that is not the feasible successor just involve updating entries in the link cost table. When a link cost $l_{s_j^i}^i$ to the feasible successor s_j^i decreases, router i just updates the distance and the feasible distance to reflect the new value $FD_j^i = D_j^i = D_{js_j^i}^i + l_{s_j^i}^i$. If $|D_j^i - RD_j^i| > \Delta D$, RD_j^i is set to the new value of D_j^i and a threshold update is broadcasted to all neighbors.

When a link cost to a feasible successor increases, router i checks to see if any other neighbor in SS_j^i still satisfies PFC and therefore can be made the new feasible successor. If PFC is not satisfied, then router i becomes active and starts a diffusing computation for destination j. Before sending out queries, the router checks if SS_j^i is non-empty, in which case it picks a neighbor m in SS_j^i as its successor. The reported distance and distance are set to the distance through m. Therefore, the queries contain the distance through m. However, the feasible distance is not changed. If SS_j^i is empty, then the reported distance, feasible distance and distance is set to the new distance through the original successor (the successor that was the feasible successor).

Once router i starts a diffusing computation for destination j, it sets its flags ST^i_{jk} to active and sends queries to all its neighbors. ST^i_{jk} remains active until a reply from k is received. Therefore, if ST^i_{jk} is set to active for any neighbor k, then router i does not forward any further queries, thus ensuring that the queries are not forwarded forever. This mechanism also helps separate different diffusing computations for the same destination.

When an active router gets replies from all its neighbors, it picks resets FD^i_j to infinity. It then picks the neighbor that satisfies PFC as the new feasible successor and sets its feasible distance, distance and reported distance equal to the distance through the new feasible successor. A router behaves differently if there have been distance increases while it was active, as explained in the following paragraph.

ROAM makes sure that, for any given destination, a router takes part in only one diffusing computation at a time. However, there might be more than one distance increase that needs to be processed while a router is active. To keep track of the multiple inputs a router may have to process, the query origin flag o^i_j is maintained by every router i for every destination j. This flag is set to 1 when a router is passive (ST^i_{jk} = passive $\forall k \in N_i$). When a router is active (ST^i_{jk} = active for some $k \in N_i$) the value of o^i_j can imply the following conditions. It must be noted that a router may get queries from any neighbor, but it becomes active only when the feasible successor no longer satisfies PFC.

- $o^i_j = 0$: Router i is the origin of the query in progress and it has experienced at least one distance increase since becoming active.
- $o^i_j = 1$: Router i is the origin of the query in progress and it has experienced no distance increase and no query from successor since becoming active.
- $o^i_j = 2$: Router i became active due to a query from a successor and it experiences a distance increase, or it is the origin of a query and receives a query from the successor after becoming active.
- $o^i_j = 3$: Router i becomes active after receiving a query from a successor and experiences no distance increases after becoming active.

When router i changes state from active to passive and $o^i_j = 1$ or 3, router i resets the value of FD^i_j to infinity. This results in router i's picking as feasible successor the neighbor that offers the shortest path. If on the other hand $o^i_j = 0$ or 2, router i retains its old FD^i_j and checks for PFC. If PFC is not satisfied, another diffusing computation is started. Before starting the diffusing computation, the values of o^i_j are changed from 0 to 1 and 2 to 3 respectively. Thus, we see that all distance increases are taken care of. A distinction is made between $o^i_j = 1$ and 3 because in the case of $o^i_j = 3$, a reply needs to be sent back to the old successor before the router becomes passive. A parallel distinction can be drawn between $o^i_j = 0$ and 2. Fig. 3 shows the states in ROAM and the transitions between them. The figure does not consider link failures and link additions, which are discussed in the next section.

E. Handling topology changes

The topology of the network can change by links going down or links coming up. When a new link comes up, it could result in partitioned sections of the network coalescing. Links going down may result in a network getting partitioned besides destroying routes. The failure of a router can be viewed as multiple links going down.

If router i detects a new neighbor k, it adds the neighbor to its routing table if the neighbor is a new destination ($D^i_k = FD^i_k = RD^i_k = \infty$, $s^i_k = k$, $o^i_j = 1$, T^i_k = present time). An entry for k is created in the distance table ($D^i_{jk} = \infty \; \forall j \in N$) and a full-state update is sent to the new neighbor. The full-state update packet contains entries for all destinations contained in router i's routing table. If router i is passive for a destination, then the entry is marked as an update, else it is marked as a query; an exception being routing entries with distance infinity which are marked as queries. The reason for this exception, given that routes are set on demand, can be explained using Fig. 4.

Consider three networks A, B and C joining. All the routers in A have the distance to destination j set to infinity. The routers in B have no entry for j and the routers in C have a finite entry for j. When the link connecting A and B comes up, if the entry for j is a simple update, then the router in B will ignore it. Therefore, even though there is a route to get to destination j which is in component C, routers in A will never be able to reach it because all of them have their distances set to infinity. Now, if the entry is a query, a diffusing search takes place in component B, at the end of which routers in A and B know the correct distance to destination j. When a router i receives a full-state update

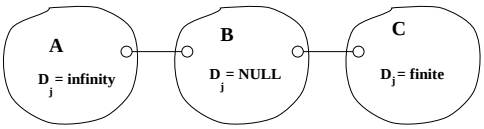


Fig. 4. Networks with different states coalescing

packet from a neighbor k, it processes each entry one by one. A query entry is processed in the manner described in sections III-C and III-D. If the entry is an update for a destination that i has no knowledge of, then i simply ignores the entry, else it records the distance ($D^i_{jk} = RD^k_j$). If the distance through the neighbor is greater than the present distance ($D^i_{jk} + l^i_k > D^i_j$), nothing is done. If the distance through the neighbor is smaller than the present distance then router i sets router k as its new feasible successor ($FD^i_j = D^i_j = D^i_{jk} + l^i_k$, $s^i_j = k$). If the change in distance is greater than a threshold value, router i sends its neighbors the new distance in updates.

If a failure of link (i, k) is detected at router i, router i sets the value D^i_{jk} to infinity for each destination j. If router i was active at the time of deletion of link (i, k), then setting ST^i_{jk} to false and D^i_{jk} to infinity mimics the behavior that would result from router i's getting a reply with distance set to infinity from router k. If i was passive and k happened to be the feasible successor, then router i becomes active and starts a diffusing computation. If (i, k) was the only link connecting router i's component and router k's component, then with the loss of link (i, k) router i loses its only successor. This results in router i's sending a query with distance set to infinity. Since this query propagates to all routers in router i's component, all of them eventually change their routing table entries to infinity, which signifies partition from the destination in router k's component.

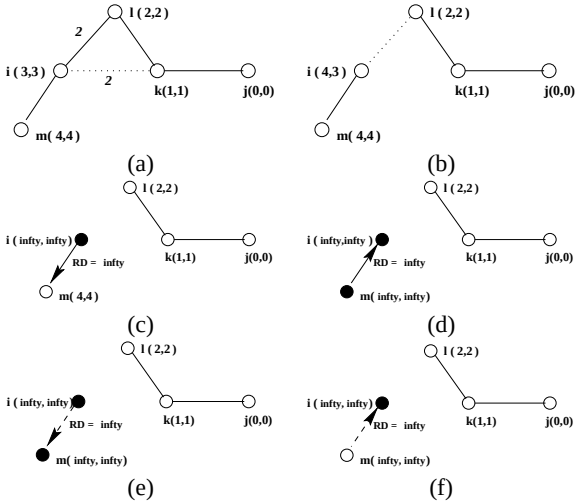
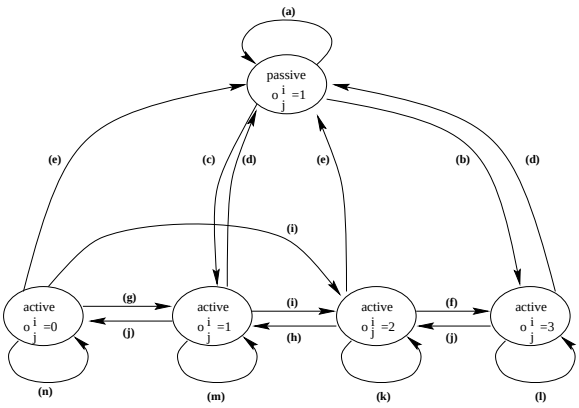


Fig. 5. Handling partitions due to link failure

Fig. 5 shows an example of a network where links go down. The



(a) delete update or input event related to nbr k which is not a successor or PFC satisfied or D^i_j = infinity and D^i_{jk} = infinity

(b) query from successor and PFC and AFC not satisfied

(c) input event other than query from successor and PFC not satisfied or query from successor with AFC satisfied and PFC not satisfied.

(d) last reply ; *action:* set FD^i_j = infinity

(e) last reply and PFC satisfied with current value of FD^i_j

(f) last reply and PFC and AFC not satisfied with current value of FD^i_j

(g) last reply and PFC not satisfied with current value of FD^i_j

(h) last reply, PFC not satisfied but AFC satisfied.

(i) query from successor

(j) increase in D^i_j

(k) input event other than last reply

(l) input event other than last reply or increase in D^i_j

(m) input event other than last reply, increase in D^i_j or query from successor

(n) input event other than last reply or query from successor.

Fig. 3. Active and Passive states in ROAM

topology and notation of the example is same as that in Fig. 2 except that the two links (i, k) and (i, l) have link costs of 2. In Fig. 5 (a) the link (i, k) fails where k is the feasible successor of i. At router i, the feasible distance is 3. Router l satisfies the feasibility condition since its distance 2 is lesser than 3 and it now offers the shortest path to the destination. Therefore, router i remains passive and changes its distance to 4. Note, however that the feasible distance does not change as it is defined to be the lowest distance value since the router became passive. In Fig. 5 (b), link (i, l) fails. Now, router i has no feasible successor. Therefore, it becomes active, sets its distance and feasible distance to infinity and sends a query to m. As shown in Fig. 5 (d), when m gets the query, it becomes active because it has no feasible successor. It also sets its distance and feasibility distance to infinity and sends a query to i. Router i sends a reply with infinite distance because it is already active. Since router m has received replies from all its neighbors, it sets its feasible distance to infinity, becomes passive and sends a reply to i. Router i then sets its feasible distance to infinity and becomes passive.

F. Deleting Routes

Routes are timestamped when they are entered into the routing table. They are also timestamped whenever data packets for the destination are seen. A timer-driven function compares the timestamp of the route to the current time at the router. If it exceeds the time threshold and if the router is not active for the destination, the route is removed from the routing table.

IV. ROAM Loop-Freedom

To prove that ROAM is correct, we need to prove that it maintains loop-free paths to all destinations and does not deadlock in any state and converges to the correct distances. Because the routes to different destinations are created and maintained independent of each other, one can prove correctness of the protocol by proving correctness for an arbitrary destination j.

The routers in N, their successors and the links from routers to their successors define a graph that we term $S_j(G)$. For the protocol to be loop-free, this graph has to be a directed acyclic graph at all times.

The complete proof of loop-freedom and correctness are presented in an extended version of this paper [11]. Here, we include a lemma which proves that ROAM is loop-free if successors are picked using PFC or AFC.

When at time $t = 0$, all the routers are initialized, they have no entries for any other destinations. The graph $S_j(G)$ consists only of all the routers in the graph with no links between them. This graph is trivially loop-free and has correct paths.

Assume that a loop $L_j(t)$ is formed for the first time at t. For a loop to be formed a router i must choose a router upstream from it in $S_j(G)$ as a successor. $L_j(t)$ is formed because a router i changes its successor from b to a due to a change in its distance $D^i_j = D^i_{jb} + l^i_b$ at time t, where b was the successor s^i_j at time t_b and $t_b < t$.

The router at the k^{th} hop at time t is $s[k, new]$ and $s[k + 1, new]$ is the successor of $s[k, new]$ at time t. The time at which $s[k, new]$ picks $s[k + 1, new]$ as its successor is denoted as $t_{s[k+1,new]}$, where $t_{s[k+1,new]} < t$. This is the last time a change was made in the routing table of $s[k, new]$ for destination j. It is seen from the definition that:

$$s^{s[k,new]}_j(t_{s[k+1,new]}) = s^{s[k,new]}_j(t)$$
$$D^{s[k,new]}_j(t_{s[k+1,new]}) = D^{s[k,new]}_j(t)$$

The time at which the last update is sent by $s[k, new]$ to its predecessor $s[k - 1, new]$ is denoted by $t_{s[k+1,old]}$. This is the last update that is sent before time t. Router $s[k, new]$'s successor at time $t_{s[k+1,old]}$ is denoted by $s[k + 1, old]$ which may or may not be the same as $s[k + 1, new]$. The times described above have the following relationship.

$$t_{s[k+1,old]} \leq t_{s[k+1,new]} \leq t$$

A path $P_{ai}(t)$ consists of the sequence of routers $\{a = s[1, new], s[2, new], ..., s[k, new], ..., i\}$, as shown in Fig. 6. If a loop $L_j(t)$ exists, then $P_{ai}(t) \subset P_{aj}(t)$. Furthermore, it also true that

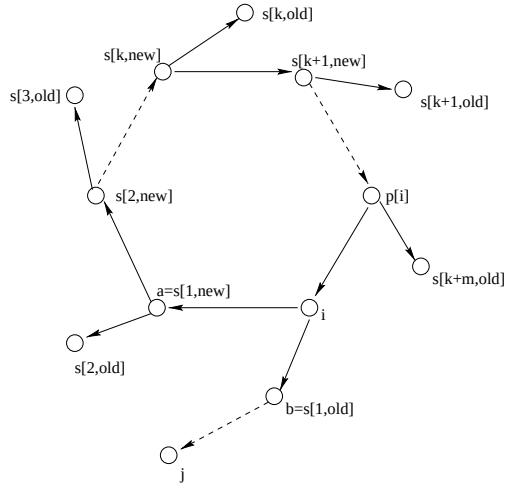


Fig. 6. Structure of possible loop caused by change of neighbor

$s^{p[i]}_j(t) = i$, $s^i_j(t_b) = b$, and $t_b < t$. By definition, $D^{*i}_j(t_i) \leq D^i_j(t_i)$ at any time t_i, and $D^{*i}_j(t_1) \leq D^{*i}_j(t_2)$ if $t_1 < t_2$.

Lemma: *If there are no diffusing computations and routers pick new successors for destination j, using PFC or AFC, then the resulting graph $S_j(G)$ is always loop-free.*

Proof: Assume that a loop $L_j(t)$ is formed for the first time at t. For a loop to be formed a router i must choose a router upstream from it in $S_j(G)$ as a successor.

A router picks a new successor only if it satisfies AFC or PFC. If either PFC or AFC have to be satisfied when a router $s[k,new] \in P_{aj}(t)$ makes router $s[k+1,new] \in P_{aj}(t)$ its successor at time $t_{s[k+1,new]}$ it must be true that

$$\begin{aligned} D^{s[k,new]}_{js[k+1,new]}(t) &= D^{s[k,new]}_{js[k+1,new]}(t_{s[k+1,new]}) \\ &< FD^{s[k,new]}_{j}(t_{s[k+1,new]}) \end{aligned}$$

Since all links costs are positive and either PFC or AFC must be satisfied by every router in $P_{ai}(t)$, we get the following inequalities while traversing it:

$$\begin{aligned} FD^i_j(t) = D^{*i}_j(t) &> D^i_{ja}(t) = D^a_j(t_{s[2,old]}) \\ D^a_j(t_{s[2,old]}) \geq D^{*a}_j(t_{s[2,old]}) &\geq D^{*a}_j(t_{s[2,new]}) \\ &= FD^a_j(t_{s[2,new]}) \\ &> D^a_{js[2,new]}(t) \\ &\vdots \\ D^{s[k-1,new]}_{js[k,new]}(t) &= D^{s[k,new]}_j(t_{s[k+1,old]}) \\ &\geq D^{*s[k,new]}_j(t_{s[k+1,old]}) \\ &\geq D^{*s[k,new]}_j(t_{s[k+1,new]}) \\ &= FD^{s[k,new]}_j(t_{s[k+1,new]}) \\ &> D^{s[k,new]}_{js[k+1,new]}(t) \\ &\vdots \\ D^{p[i]}_{ji}(t) &= D^i_j(t_b) \geq D^{*i}_j(t) = FD^i_j(t). \end{aligned}$$

The above set of inequalities leads to the erroneous conclusion that $FD^i_j(t) > FD^i_j(t)$. Therefore, it follows that no loop can be formed in $S_j(G)$ if the PFC and AFC are used while picking a new successor. □

V. Complexity

The performance of ROAM can be measured in terms of the time and communication overhead required to get routing tables to converge and have loop-free paths to the destinations. Actual time is hard to predict because it involves predicting varying inter-router communication time and other delays associated with queuing, etc. Consequently, we assume that the protocols behave synchronously, which means that all actions are taken by the routers in discrete steps. A router receives its inputs, processes them, makes changes to its routing tables and sends updates all in the same step. The neighboring routers receive the updates in the next step. We start measuring the number of steps and messages after a single topological change. This change could be a link failure, link addition or a link cost change. The neighboring router discovers the change in the first step. During the last step, at least one router receives and processes updates from a neighbor, after which all routing tables are correct and no more updates need to be sent till the next topological change. *Time complexity* measures the number of steps it takes for this process and *communication complexity* measures the number of messages it takes.

In ROAM, a router searches for a destination if the destination is not already in the routing tables. This involves sending a query with an infinite distance for the destination. The query is broadcast to all neighbors. Each neighbor that gets the query checks to see if it has a routing table entry for the destination. If it does not, then the neighbor becomes active and sends a query with infinite distance to all its neighbors which includes the one that sent it the original query. A router that is already active and receives a query does not send any more queries. Thus, one can see that a search query cannot be sent over a link more than twice. Therefore, the communication complexity is $O(|E|)$, where $|E|$ is the number of edges in the network. The time complexity is $O(d)$ where d is the diameter of the network.

After a single link failure or link-cost increase, the time complexity is the same as the Jaffe-Moss algorithm [5]. In the worst case, all routers upstream of the destination must freeze their routing table entries for the destination. Therefore, the time complexity is $O(x)$, where x is the number of routers affected by the routing table change. The communication complexity is $O(6Dx)$, where D is the maximum degree of the router.

Any router that receives information reporting a distance decrease will always be able to find a feasible successor. Updates are only sent if the distance changes by more than the threshold. Therefore, link additions can at best have no reaction, at worst have a message complexity of $O(2Dx)$ and a time complexity of $O(l)$, where l is the longest path to a destination.

VI. Summary and Conclusions

We have presented ROAM, the first on-demand routing algorithm that provides multiple loop-free paths based solely on distances to destinations, without the need for complete path information, sequence numbers refreshed periodically, or time synchronization. We have shown elsewhere that ROAM is loop-free at every instant and that it converges within a finite time. We also introduced the search-to-infinity problem and eliminated its occurrence in ROAM, such that sources do not send repeated flood searches in the event of destinations being unreachable. Given its performance, ROAM is very applicable to wired networks, where multiple paths to the Internet are provided on demand using ROAM, and wireless networks with static nodes or nodes with limited mobility.

References

[1] D. Bertsekas and R. Gallager. *Data Networks.* Prentice-Hall Inc., 2nd edition, 1992.

[2] E. M. Gafni and D.P. Bertsekas. Distributed Algorithms for Generating Loop-Free Routes in Networks with Frequently Changing Topology. *IEEE Trans. Commn.*, COM-29(1):11–18, January 1981.

[3] J. J. Garcia-Luna-Aceves. Loop-Free Routing Using Diffusing Computations. *IEEE/ACM Transactions on Networking*, 1(1):130–141, Feb 1993.

[4] C. Huitema. *Routing in the Internet*, chapter 6, pages 145–148. Prentice Hall PTR, 1995.

[5] J. M. Jaffe and F. H. Moss. A Responsive Distributed Routing Algorithm for Computer Networks. *IEEE Trans. Commun.*, COM-30(7):1758–1762, July 1982.

[6] D. B. Johnson and D. A. Maltz. Dynamic Source Routing in Ad-Hoc Wireless Networks. *Mobile Computing*, 1994.

[7] J. Ju and V.O.K. Li. An Optimal Topology-Transparent Method in Multihop Packet Radio Networks. *IEEE/ACM Trans. Networking*, 6(3), June 1998.

[8] J. Moy. *OSPF Version 2*, 1991. RFC 1247.

[9] V. D. Park and M. S. Corson. A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. In *Proc. IEEE INFOCOM'97*, Kobe, Japan, April 1997.

[10] C. E. Perkins. Ad Hoc On-Demand Distance Vector (AODV) Routing. Internet Draft–Mobile Ad hoc NETworking (MANET) Working Group of the Internet Engineering Task Force (IETF). To be considered Work in Progress., November 1997.

[11] J. Raju and J.J. Garcia-Luna-Aceves. ROAM: A New Approach to On-Demand Loop-Free Multipath Routing. http://www.cse.ucsc.edu/~jyoti/publications.html.

[12] T. Shepard. A Channel Access Scheme for Large Dense Packet Radio Networks. In *Proc. ACM SIGCOMM*, Aug 96.

[13] C. Zhu and S. Corson. A Five-Phase Reservation Protocol (FPRP) for Mobile Ad Hoc Networks. In *Proc. IEEE Infocom*, 98.